Truly Selective Refinement of Progressive Meshes

Junho Kim
victor@postech.ac.kr

Seungyong Lee
leesy@postech.ac.kr

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, 790-784, Korea
http://graphics.postech.ac.kr/

Abstract

This paper presents a novel selective refinement scheme of progressive meshes. In previous schemes, topology information in the neighborhood of a collapsed edge is stored in the analysis phase. A vertex split or edge collapse transformation is possible in the synthesis phase only if the configuration of neighborhood vertices in the current mesh corresponds to the stored topology information. In contrast, the proposed scheme makes it possible to apply a vertex split or an edge collapse to any selected vertex or edge in the current mesh without a precondition. Our main observation is that the concept of a *dual piece* can be used to clearly enumerate and visualize the set of all possible selectively refined meshes for a given mesh. Our refinement scheme is truly selective in the sense that each vertex split or edge collapse can be performed without incurring additional vertex split and/or edge collapse transformations.

Key words: Selective refinement, progressive mesh, dual piece, progressive transitive mesh space, topological detail, cut vertex.

1 Introduction

With the development of 3D scanners, very large-scale polygonal meshes are widely used to represent highly detailed models of complicated objects. To achieve fast visualization and processing of large polygonal meshes, much research has been done on mesh simplification [23, 21, 11, 18, 4, 15] and the multiresolution representation of meshes [2, 8, 20, 6]. Multiresolution mesh representations offer continuous levels-of-detail meshes by using mesh simplification operators and their inverse operators. Various simplification operators have been introduced, such as vertex unification, vertex removal & retriangulation, edge collapse, and face collapse. To accelerate the rendering further, or to allow the locally adaptive processing of a large-scale mesh, several adaptive refinement schemes have been proposed for multiresolution representations [8, 24, 9, 25, 3].

A *progressive mesh* is the multiresolution representation of an irregular mesh based on edge collapse and vertex split transformations [8]. To build the progressive mesh representation, a given mesh is reduced to a base mesh by a sequence of edge collapse transformations. Similar to wavelet analysis, each edge collapse transformation reserves *detail information* to recover the previous mesh. This progressive analysis of a given mesh is generally performed during an off-line preprocessing, which we call the *analysis phase*. Given a base mesh and the set of detail information, continuous sequence of levels-of-detail meshes can be generated with successive vertex split transformations. Furthermore, by applying a chosen set of vertex split transformations, a progressive mesh can be selectively refined, depending on certain criteria such as rendering parameters. We call this run-time processing of a progressive mesh to construct a selectively refined mesh the *synthesis phase*.

Previous selective refinement schemes of progressive meshes reserve a 1-ring neighbor configuration of an edge as the topological detail information of an edge collapse transformation in the analysis phase [8, 24, 9, 3]. In the synthesis phase, a vertex split or edge collapse transformation is possible only if the 1-ring neighborhood in the current mesh corresponds to the stored information. If this precondition is not met, additional vertex splits and/or edge collapses are incrementally applied to the neighborhood until the transformed configuration matches the stored topology. These *incremental* approaches, however, have the fundamental limitation that the resolution of a refined mesh must change gradually.

This paper proposes a *truly* selective refinement scheme for progressive meshes. With the scheme, a vertex split or an edge collapse transformation can always be applied to a selectively refined mesh without preconditions. In other words, when a vertex is split and an edge is collapsed selectively, this does not incur other vertex or edge transformation, regardless of the configuration of the neighborhood. Further, the selectively refined mesh is updated locally only in the vicinity of a split vertex or collapsed edge.

Our main observation is that the concept of the *dual piece* can be used to clearly enumerate and visualize the set of all possible selectively refined meshes, when the set is fixed after the analysis phase. We define the set as the *progressive transitive mesh space*. In the progressive transitive mesh space, a selectively refined mesh is an element, and a vertex split or edge collapse transformation is an operator which maps an element in the space to an adjacent element. We provide new definitions of vertex split and edge collapse transformations, which make it possible to traverse the whole progressive transitive mesh space.

Since the selective refinement of progressive meshes was investigated to accelerate the rendering of a complicated mesh, research in this area was concerned with the design of efficient visual error metrics such as back-face culling and screen-space

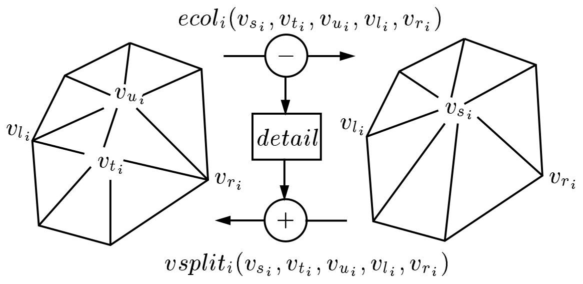

$ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$
$-$
$detail$
$+$
v_{u_i}
v_{l_i}
v_{t_i}
v_{r_i}
v_{s_i}
v_{l_i}
v_{r_i}
$vsplit_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$

Figure 1: Edge collapse and vertex split transformations

error. Several elegant methods for fast visual error measurement have been proposed [16, 9, 10]. In this paper, we do not consider a visual error metric because we do not intend to present a new view-dependent rendering framework for progressive meshes.

In this paper, we refer a progressive mesh to any multiresolution representation of a mesh based on edge collapse and vertex split transformations. That is, the proposed selective refinement scheme can be applied to any mesh, once the vertex hierarchy has been constructed through edge collapse sequences, regardless of the strategies for selecting the collapsed edge and determining the position of the new vertex.

The remainder of this paper is organized as follows. In Section 2, we review previous work. We analyze the progressive transitive mesh space in Section 3 and propose the truly selective refinement scheme of progressive meshes in Section 4. We present experimental results in Section 5. Section 6 concludes the paper.

2 Previous Work

2.1 Progressive Meshes

Hoppe [8] introduced progressive mesh representation based on edge collapse and vertex split transformations. In the analysis phase, a base mesh M^0 and details are obtained from the given mesh $\hat{M}$ by n times successive edge collapse transformations.

$$\hat{M} = M^n \xrightarrow{ecol_{n-1}} M^{n-1} \xrightarrow{ecol_{n-2}} \cdots \xrightarrow{ecol_0} M^0$$

In each $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ transformation from M^{i+1} to M^i, detail information d_i is reserved for recovering M^{i+1} from M^i in the synthesis phase. Each detail d_i generally consists of information about topology (i.e., connectivity), geometry, and other mesh data (e.g., texture coordinates, material id, etc.) (see Figure 1). The i-th resolution mesh M^i can simply be reconstructed by applying the vertex split sequence, $\{vsplit_0, vsplit_1, ..., vsplit_{i-1}\}$, to the base mesh M^0.

$$M^0 \underset{ecol_0}{\overset{vsplit_0}{\rightleftharpoons}} M^1 \underset{ecol_1}{\overset{vsplit_1}{\rightleftharpoons}} \cdots \underset{ecol_{n-1}}{\overset{vsplit_{n-1}}{\rightleftharpoons}} M^n$$

This sequential refinement scheme of progressive meshes has been successful in mesh transformation, mesh compression, and mesh editing [8, 14, 7, 17].

2.2 Selective Refinement of Progressive Meshes

In selective refinement of progressive meshes, a chosen subsequence of $\{vsplit_0, vsplit_1, ..., vsplit_{n-1}\}$ is applied to the base mesh M^0 [9]. Equivalently, any vertex split or edge collapse transformation related to a vertex or an edge in the current mesh can be performed in the synthesis phase. In this case, however, we confront vexing dependency problems, which are inherited from the correlation among the topological details. For example, it is not clear how we can split the vertex v_{s_i} into two vertices v_{t_i} and v_{u_i} when v_{l_i} and/or v_{r_i} are not present in the current mesh. To resolve this dependency problem, several selective refinement schemes have been proposed [8, 24, 9, 3].

Hoppe [8] suggested two conditions of a legal $vsplit$ transformation for selective refinement of progressive meshes. The first condition of $vsplit(v_{s_i}, \cdots)$ is the activeness of the vertices v_{l_i} and v_{r_i}. If v_{l_i} or v_{r_i} is absent from the current mesh, additional $vsplit$ transformations are invoked to activate the absent vertices. He also indicated that if the current mesh contains active ancestors of v_{l_i} and v_{r_i}, $vsplit(v_{s_i}, \cdots)$ with these active ancestor vertices is legal. However, he did not clearly analyze the $vsplit$ case when the descendent of v_{l_i} or v_{r_i} is active.

Xia and Varshney [24] introduced an incremental selective refinement scheme for progressive meshes. After each halfedge collapse, $e_{s_i t_i} \rightarrow v_{s_i}$, 1-ring neighbor vertices of v_{s_i} are reserved as topological details in the analysis phase. In the synthesis phase, both $vsplit(v_{s_i}, v_{t_i}, \cdots)$ and $ecol(v_{s_i}, v_{t_i}, \cdots)$ are valid only when the current neighbor of v_{s_i} is identical to the reserved topological details. If any vertex of the topological details does not exist in the current mesh, additional $vsplit$ and/or $ecol$ transformations are required to activate the vertex prior to $vsplit(v_{s_i}, v_{t_i}, \cdots)$ or $ecol(v_{s_i}, v_{t_i}, \cdots)$ transformation. Dependency among the transformations is checked using the merge tree structure.

Hoppe [9] noticed that the full set of the neighborhood vertices is not necessary. Instead, he stored four faces $(f_{n0}, f_{n1}, f_{n2}, f_{n3})$ adjacent to the collapsed faces (f_l, f_r) as topological details in the analysis phase (cf. Figure 2 in [9]). In the synthesis phase, the two vertices v_{ll}, v_{rr} for the $vsplit(v_{s_i}, v_{t_i}, v_{u_i}, v_{ll}, v_{rr})$ and $ecol(v_{s_i}, v_{t_i}, v_{u_i}, v_{ll}, v_{rr})$ transformations are obtained from the configuration of the four faces. Since three vertices of a triangle dynamically change in the selective refinement process, the two vertices v_{ll} and v_{rr} are generally not the same as vertices v_{l_i} and v_{r_i}, which were the two opposite vertices of the edge $e_{t_i u_i}$ in the analysis phase.

El-sana and Varshney [3] extended the condition of Xia and Varshney [24] to implicit representation and also made the genus-change possible with the vertex pair collapse transformation. In the analysis phase, increasing vertex-ids are given to the vertices produced from vertex pair collapses, and therefore the vertex-id of each vertex retains the generation order of the vertex. To reserve topological details in an implicit fashion, each vertex reserves min/max vertex-ids among its 1-ring neighbor vertices in the analysis phase. In the synthesis phase, the min/max vertex-ids are used for a validity test of a $vsplit$ or $ecol$ transformation.

2.3 Limitation of Previous Schemes

Table 1 summarizes the selective refinement schemes for progressive meshes. Although topological details differ from each other, previous schemes share a common factor. That is, the topological details reserved in the analysis phase are the entities

Author	Topological details	Year
Hoppe [8]	v_{l_i} and v_{r_i} on M^i	1996
Xia et al. [24]	1-ring neighbor vertices of v_{s_i} on M^i	1996
Hoppe [9]	four adjacent faces on M^i	1997
El-sana et al. [3]	min/max vertex-ids among the 1-ring neighbor vertices of v_{s_i} on M^i	1999
Our scheme	$\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ on $\hat{M}$	2001

Table 1: Taxonomy of the topological details for selective refinement schemes

on the simplified mesh M^i, not on the original mesh $\hat{M}$. When we intend to perform $vsplit(v_{s_i}, \cdots)$ or $ecol(v_{s_i}, \cdots)$ in the synthesis phase, the topological details may not be available in the current mesh because a selectively refined mesh changes dynamically. To restore the required topological details into the current mesh, additional vertex split and/or edge collapse transformations around the vertex v_{s_i} are inevitable. This restriction prohibits an abrupt change in levels-of-detail between different parts of a selectively refined mesh.

The scheme provided by Xia and Varshney acts like a quadtree refinement, as Hoppe [9] pointed out. The scheme of El-sana and Varshney has the same limitation because it extend the scheme of Xia and Varshney with implicit representation. The four-face condition of Hoppe [9] is more flexible than previous schemes but is still dependent on the activeness of the four faces. This activeness condition could produce a long dependency chain. For example, to satisfy the condition for a vertex split transformation, the current mesh may be refined up to the original mesh in the worst case (see Figure 2).

In contrast to previous work, we represent the topological details by the vertices in the original mesh $\hat{M}$. With this representation, additional vertex split and/or edge collapse transformations are not necessary for $vsplit(v_{s_i}, \cdots)$ or $ecol(v_{s_i}, \cdots)$.

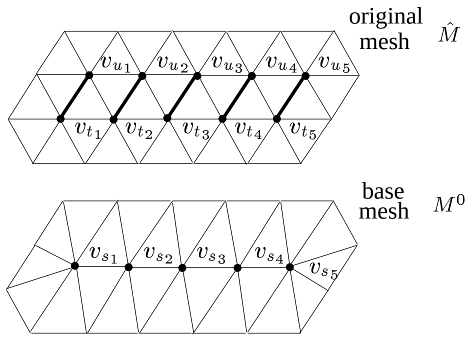
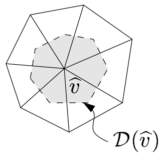
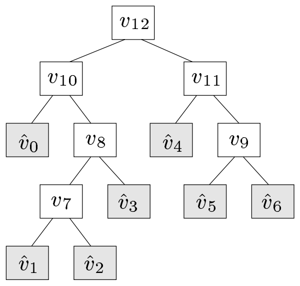


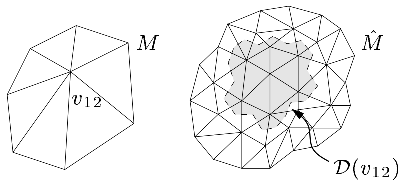
Figure 2: Worst case of the four-face scheme of Hoppe [9]: Assume that the base mesh M^0 is obtained by the edge collapse sequence, $ecol_5$, $ecol_4$, $\cdots$, $ecol_1$. To split the vertex v_{s_5}, M^0 must be refined to the original mesh $\hat{M}$.



(a) a fundamental dual piece



(b) a subtree in the vertex hierarchy: $\hat{v}_0, \hat{v}_1, \cdots, \hat{v}_6$ are the vertices in the original mesh $\hat{M}$.



(c) the dual piece of v_{12}

Figure 3: Dual piece of a vertex in the vertex hierarchy

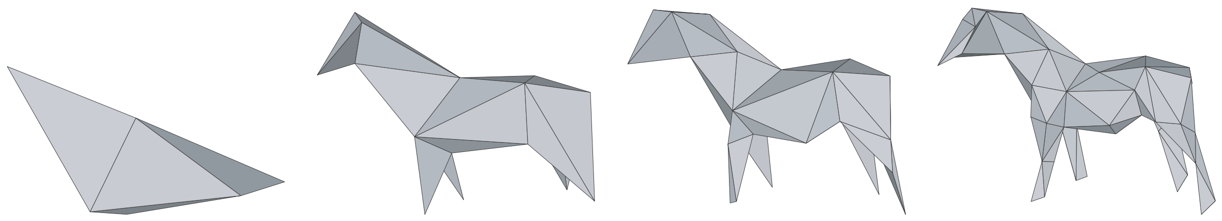
3 Transitive Space of Progressive Meshes

In this section, we introduce the *progressive transitive mesh space* of a given mesh $\hat{M}$. The progressive transitive mesh space $\mathcal{S}_H(\hat{M})$ consists of all selectively refined meshes that can be obtained from the base mesh M^0. Note that the space $\mathcal{S}_H(\hat{M})$ is fixed when the vertex hierarchy H has been constructed by a simplification process. Although the given mesh $\hat{M}$ is the same, $\mathcal{S}_H(\hat{M})$ is changed if $\hat{M}$ is simplified in a different way. With the analysis of the progressive transitive mesh space, we obtain important clues to the design of more effective vertex split and edge collapse transformations. We use the *hat* notation to denote an entity related to the given mesh $\hat{M}$; for example, $\hat{v}$ denotes a vertex in $\hat{M}$.

3.1 Hierarchical Partitioning Property

A vertex hierarchy is constructed by the sequence of edge collapses in the analysis phase [9]. In each edge collapse $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$, the edge $e_{t_i u_i}$ is collapsed to a new vertex v_{s_i} and the *parent-children relationship* is established between v_{s_i} and v_{t_i}/v_{u_i}. After n successive edge collapse transformations from the given mesh $\hat{M}$, we obtain a base mesh M^0 and a vertex hierarchy H, which is a forest structure. The root nodes of H are the vertices of the base mesh M^0 and all leaf nodes of H are the vertices of the given mesh $\hat{M}$.

To explain the hidden linkage between a selectively refined


(a) M^4 (b) M^{16} (c) M^{32} (d) M^{64}

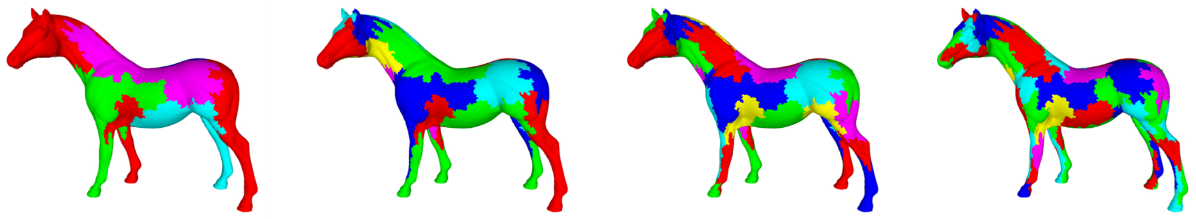
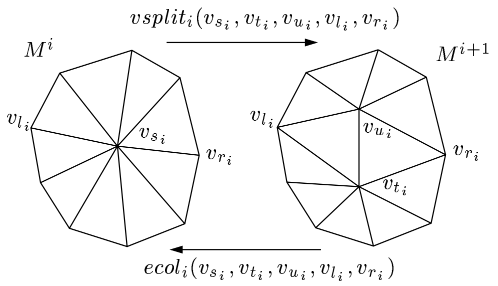

(e) dual pieces of M^4 (f) dual pieces of M^{16} (g) dual pieces of M^{32} (h) dual pieces of M^{64}

Figure 4: Progressive meshes and the corresponding dual pieces overlaid on the original mesh


$vsplit_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$
M^i
M^{i+1}
v_{l_i}
v_{s_i}
v_{r_i}
v_{l_i}
v_{u_i}
v_{r_i}
v_{t_i}
$ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$
(a) meshes M^i and M^{i+1}

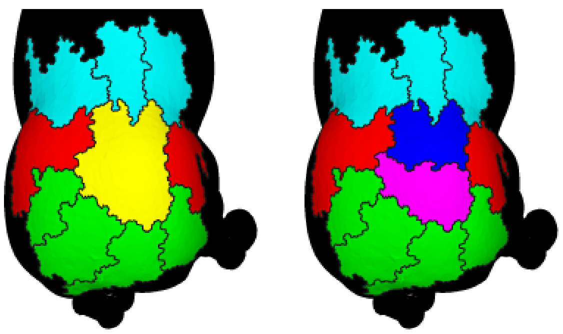

(b) corresponding dual pieces on the original mesh

Figure 5: Dual perspective of $vsplit$ and $ecol$ transformations

mesh M and the finest resolution mesh $\hat{M}$, we first define the *fundamental dual piece* of a vertex $\hat{v}$ in $\hat{M}$. The dual of a planar graph is constructed by assigning a dual vertex to each face of the graph and connecting a pair of dual vertices with a dual edge if the corresponding faces share an edge in the graph. In this fashion, the fundamental dual piece of $\hat{v}$ is defined as the closed region over a given mesh $\hat{M}$, and surrounded by dual edges that connect the dual vertices corresponding to the faces adjacent to $\hat{v}$ (see Figure 3(a)). We also define the *dual piece* $\mathcal{D}(v)$ of a vertex v in the vertex hierarchy H as the union of fundamental dual pieces of all leaf nodes in the subtree of H whose root is v (see Figure 3(b) and (c)). Note that the dual piece $\mathcal{D}(v)$ of a vertex v in H is always defined over a given mesh $\hat{M}$.

For the dual pieces of vertices in the vertex hierarchy H, the following properties hold.

- $\mathcal{D}(v_{s_i}) = \mathcal{D}(v_{t_i}) \cup \mathcal{D}(v_{u_i})$ and $\mathcal{D}(v_{t_i}) \cap \mathcal{D}(v_{u_i}) = \emptyset$, for all i.
- $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$ are adjacent to each other, for all i.
- $\mathcal{D}(v_q) \subset \mathcal{D}(v_p)$ if v_p is an ancestor of v_q in H.
- $\mathcal{D}(v_p) \cap \mathcal{D}(v_q) = \emptyset$ if v_p and v_q have no ancestor-descendent relationship in H.

Figure 4 shows several sequential levels-of-detail versions M^i of a horse mesh and the dual pieces of vertices in M^i overlaid on $\hat{M}$. Note that an edge between a pair of vertices in M^i emerges as an adjacency between the corresponding dual pieces on $\hat{M}$.

For a selectively refined mesh M, the set of vertices V in M has the following properties: i) any pair of vertices in V has no ancestor-descendent relationship, and ii) a leaf node in H has only one ancestor node in V. So, we observe that the dual pieces of vertices in V cover the original mesh $\hat{M}$ without overlaps and holes. In other words, the dual pieces from M partition the surface of $\hat{M}$. Further, the partitioning becomes locally finer when a vertex in M is split.

3.2 Progressive Transitive Mesh Space

With the hierarchical partitioning property, the progressive transitive mesh space $\mathcal{S}_H(\hat{M})$ of a given mesh $\hat{M}$ can be well defined. Consider a set of vertices V in the vertex hierarchy H. We define the vertex set V to be *valid* if it satisfies the following properties.

- The dual pieces of V cover the original mesh $\hat{M}$ without overlaps and holes.
- The dual pieces of V are simply connected. That is, any two adjacent dual pieces share only one portion of their boundaries.[1]

For a valid vertex set V in H, we can easily construct the corresponding selectively refined mesh M. The vertices of M is naturally identical with V. The edges of M are derived from the adjacency among the dual pieces of V. Hence, the progressive transitive mesh space $\mathcal{S}_H(\hat{M})$ is isomorphic to the set of valid vertex sets in H.

Moreover, we can consider the selective refinement of a progressive mesh as the traversal of the progressive transitive mesh space with vertex split and edge collapse transformations. Let M be the current selectively refined mesh defined by a valid vertex set V. Then, $vsplit(v_{s_i}, v_{t_i}, v_{u_i}, \cdots)$ is expected to transform the mesh M to the selectively refined mesh defined by $(V - \{v_{s_i}\}) \cup \{v_{t_i}, v_{u_i}\}$. Similarly, $ecol(v_{s_i}, v_{t_i}, v_{u_i}, \cdots)$ should produce the selectively refined mesh defined by $(V - \{v_{t_i}, v_{u_i}\}) \cup \{v_{s_i}\}$ from the mesh M. However, all previous schemes do not provide this *truly* selective refinement property with vertex split and edge collapse transformations, as mentioned in Section 2.3.

3.3 Dual Perspective of Progressive Mesh Transformation

From this dual perspective, we can see that it is feasible to design vertex split and edge collapse transformations with the truly selective refinement property. Figure 5 shows the dual perspective of $vsplit_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ and $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$. In the dual space, a $vsplit$ transformation corresponds to re-tiling a piece $\mathcal{D}(v_{s_i})$ with two adjacent pieces $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. Similarly, with an $ecol$ transformation, two adjacent dual pieces are merged into one. Note that the dual pieces of 1-ring neighbor vertices of v_{s_i} are *invariant* under $vsplit_i(v_{s_i}, v_{t_i}, v_{r_i}, v_{l_i}, v_{r_i})$ or $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$. This implies that it is possible to split a vertex or collapse an edge in a selectively refined mesh without affecting the neighbor vertices.

As shown in Figure 5(a), two vertices v_{l_i} and v_{r_i} determine the connection among v_{t_i}/v_{u_i} and the 1-ring neighbor vertices $N(v_{s_i})$ of v_{s_i} after $vsplit_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$. In the dual perspective of Figure 5(b), v_{l_i} and v_{r_i} are only the vertices among $N(v_{s_i})$ whose dual pieces are consecutively adjacent to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$.

Suppose that we wish to split vertex v_{s_i} when it resides in a selectively refined mesh M. Since the current 1-ring neighbor of v_{s_i} is dynamically changed in the synthesis phase, it may differ from the 1-ring neighbor of v_{s_i} at the time when $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ was performed at the analysis phase. Therefore, to split v_{s_i} in M, we must find two vertices among $N(v_{s_i})$ which play the roles of v_{l_i} and v_{r_i} in $vsplit_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$. We designate two such vertices as the *cut vertices* of v_{s_i} in M.

[1]The simply connectedness constraint is inherited from the fact that $vsplit$ and $ecol$ transformations can only deal with simply connected meshes. See more details in Sec 4.2

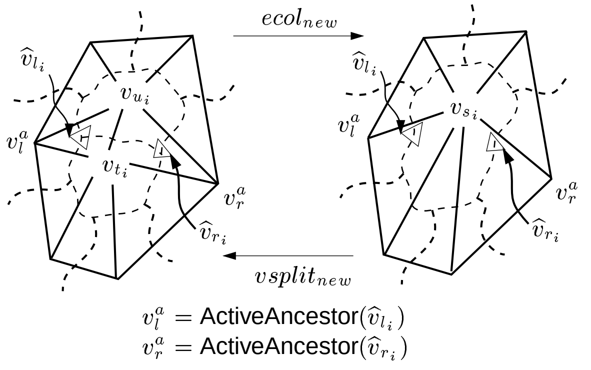


Figure 6: New definitions of $vsplit$ and $ecol$ transformations

From the dual perspective, we can see that the cut vertices of v_{s_i} are the vertices in the current 1-ring neighbor whose dual pieces are consecutively adjacent to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$, and that they always exist. However, in the mesh space, it is not clear how to locate the cut vertices in the current 1-ring neighbor of v_{s_i}.

4 Truly Selective Refinement Scheme

In this section, we propose a truly selective refinement scheme of progressive meshes, based on the dual perspective presented in Section 3. We first design vertex split and edge collapse transformations with the truly selective refinement property. Next, we explain how to reserve the required topological details in the analysis phase. We also explain the operation of the transformations using the topological details in the synthesis phase. Finally, we comment on degenerate cases.

4.1 Design of New Transformations

The key to locating the cut vertices of v_{s_i} in the current neighborhood is to use the hierarchical partitioning property of dual pieces. Recall that the dual pieces of the cut vertices of v_{s_i} are consecutively adjacent to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. Let $\hat{v}$ be a *leaf node* in the vertex hierarchy H whose fundamental dual piece $\mathcal{D}(\hat{v})$ has adjacency to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. Then, the dual piece $\mathcal{D}(v^a)$ of any active ancestor v^a of $\hat{v}$ has the same adjacency because $\mathcal{D}(\hat{v}) \subset \mathcal{D}(v^a)$. We define the *fundamental cut vertices* of the vertex v_{s_i} as the two leaf nodes $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ in H such that $\mathcal{D}(\hat{v}_{l_i})$ and $\mathcal{D}(\hat{v}_{r_i})$ are adjacent to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. We can determine the cut vertices of v_{s_i} in the current neighborhood of M by finding the active ancestors of the fundamental cut vertices $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$.

Now we introduce new definitions of vertex split and edge collapse transformations for truly selective refinement of progressive meshes. For $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ in the analysis phase, we reserve the fundamental cut vertices $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ of v_{s_i} as topological details. In the synthesis phase, a vertex split or edge collapse transformation about v_{s_i} is performed with the cut vertices of v_{s_i} that are dynamically determined by finding the active ancestors of $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$. We define the $vsplit_{new}$ and $ecol_{new}$ transformations in the synthesis phase as follows (see Figure 6).

$$\begin{aligned} & vsplit_{new}(v_{s_i}, v_{t_i}, v_{u_i}, \hat{v}_{l_i}, \hat{v}_{r_i}) \\ & \qquad = vsplit(v_{s_i}, v_{t_i}, v_{u_i}, v_l^a, v_r^a) \\ & ecol_{new}(v_{s_i}, v_{t_i}, v_{u_i}, \hat{v}_{l_i}, \hat{v}_{r_i}) \\ & \qquad = ecol(v_{s_i}, v_{t_i}, v_{u_i}, v_l^a, v_r^a), \end{aligned}$$

where

$$\begin{aligned} v_l^a &= \mathsf{ActiveAncestor}(\hat{v}_{l_i}) \\ v_r^a &= \mathsf{ActiveAncestor}(\hat{v}_{r_i}). \end{aligned}$$

4.2 Analysis Phase

Now we show how to locate the fundamental cut vertices $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ for each $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ in the analysis phase. The basic idea is to use the correspondence of the faces between a simplified mesh M^i and the original mesh $\hat{M}$. Let $f = \triangle(\hat{v}_p, \hat{v}_q, \hat{v}_r)$ be a triangle in $\hat{M}$. As edge collapse transformations are performed, face f is relabeled with active ancestors of $\hat{v}_p$, $\hat{v}_q$, and $\hat{v}_r$. Therefore, face f remains alive in M^i only when all three vertices of f have different ancestors.

Let $f_l = \triangle(v_{t_i}, v_{u_i}, v_{l_i})$ and $f_r = \triangle(v_{u_i}, v_{t_i}, v_{r_i})$ be the two faces collapsed by $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$. Suppose that $\triangle(\hat{v}_{tl_i}, \hat{v}_{ul_i}, \hat{v}_{l_i})$ and $\triangle(\hat{v}_{ur_i}, \hat{v}_{tr_i}, \hat{v}_{r_i})$ are the faces in $\hat{M}$ that correspond to f_l and f_r, respectively (see Figure 7). It follows that $v_{l_i} = \mathsf{ActiveAncestor}(\hat{v}_{l_i})$ and similar relationships for other pairs of vertices. The fundamental dual piece of $\hat{v}_{l_i}$ is adjacent to those of $\hat{v}_{tl_i}$ and $\hat{v}_{ul_i}$ because $\hat{v}_{tl_i}$, $\hat{v}_{ul_i}$, and $\hat{v}_{l_i}$ are the vertices of a triangle in $\hat{M}$. From that $\mathcal{D}(\hat{v}_{tl_i}) \subset \mathcal{D}(v_{t_i})$ and $\mathcal{D}(\hat{v}_{ul_i}) \subset \mathcal{D}(v_{u_i})$, $\mathcal{D}(\hat{v}_{l_i})$ has adjacency to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. Hence, $\hat{v}_{l_i}$ is a fundamental cut vertices of v_{s_i}. Similarly, $\hat{v}_{r_i}$ is another fundamental cut vertex of v_{s_i}.

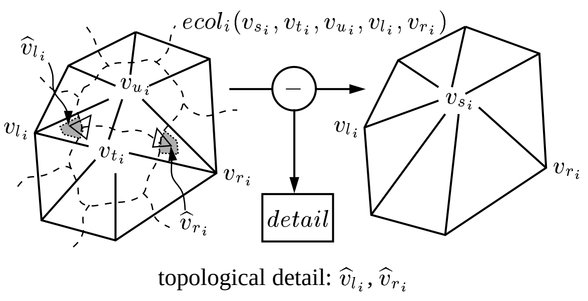


Figure 7: New topological details reserved in the analysis phase: The shaded regions denote the dual pieces of the fundamental cut vertices of v_s.

In the analysis phase, our data structure for a triangle face contains three current vertices $\{v_p, v_q, v_r\}$ in M^i and their corresponding vertices $\{\hat{v}_p, \hat{v}_q, \hat{v}_r\}$ in the original mesh $\hat{M}$. When we perform an $ecol_i(v_{s_i}, v_{t_i}, v_{u_i}, v_{l_i}, v_{r_i})$ transformation, we reserve $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ obtained from the two triangles, $\triangle(v_{t_i}, v_{u_i}, v_{l_i})$ and $\triangle(v_{u_i}, v_{t_i}, v_{r_i})$, as topological details of v_{s_i}. If $e_{t_i u_i}$ is a boundary edge, we reserve only one vertex, either $\hat{v}_{l_i}$ or $\hat{v}_{r_i}$. With this approach, additional $\Theta(3n)$ storage is required to store the original vertices in each triangle, where n is the number of triangles in $\hat{M}$. However, the additional storage is required only in the analysis phase and is not necessary in the synthesis phase.

4.3 Synthesis Phase

Let $e_{t_i u_i}$ be an edge in a selectively refined mesh M. To apply $ecol_{new}(v_{s_i}, v_{t_i}, v_{u_i}, \hat{v}_{l_i}, \hat{v}_{r_i})$ to the edge, we must know the cut vertices v_l^a and v_r^a of v_{s_i}, which are the active ancestors of $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$, respectively. In this case, however, we can immediately locate the cut vertices, v_l^a and v_r^a, among the neighborhood of v_{t_i}/v_{u_i} without referring to $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$. Since the opposite vertices of $e_{t_i u_i}$ are adjacent to both v_{t_i} and v_{u_i}, their dual pieces are also adjacent to both $\mathcal{D}(v_{t_i})$ and $\mathcal{D}(v_{u_i})$. Therefore, the opposite vertices of $e_{t_i u_i}$ are always the active ancestors of $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$.

Unlike an $ecol_{new}$ transformation, in the case of $vsplit_{new}$, it is difficult to find the cut vertices v_l^a and v_r^a without referring to $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$. A straightforward approach to find v_l^a and v_r^a from $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$ is to use the ActiveAncestor() procedure, which climbs up the vertex hierarchy H from a leaf node until an active node is reached. With this approach, a $vsplit_{new}$ transformation takes $O(\log n)$ time in the worst case even when the vertex hierarchy is balanced, where n is the number of vertices in the given mesh $\hat{M}$.

To speed up $vsplit_{new}$ transformations, we index each node in the vertex hierarchy H with the <*tree-id, node-id*> notation before starting the synthesis phase. The nodes in the same tree of H have the same tree-id. In a tree, we assign a proper node-id to each node in a similar manner to the array implementation of a heap [13]. We design an $\mathsf{IsAncestor}(v_i, v_j)$ procedure to test whether v_i is an ancestor of v_j or not by using a binary shift operation. When the *tree-id*s of v_i and v_j are different, the procedure simply returns false. Otherwise, it shifts the *node-id* of v_j to the right by the amount of the level difference between v_i and v_j, and then compares the result with *node-id* of v_i. With this simple test, we can determine whether v_i is an ancestor of v_j or not within $O(1)$ time. In our experiments, a 64-bit unsigned integer was sufficient for the <*tree-id, node-id*> representation.

To locate the cut vertices v_l^a and v_r^a from $\hat{v}_{l_i}$ and $\hat{v}_{r_i}$, we test each vertex v_n in the current 1-ring neighborhood $N(v_{s_i})$ of v_{s_i} with $\mathsf{IsAncestor}(v_n, \hat{v}_{l_i})$ and $\mathsf{IsAncestor}(v_n, \hat{v}_{r_i})$. Then, we determine the cut vertices v_l^a and v_r^a among $N(v_{s_i})$ in $O(k)$ time, where k is the number of vertices in $N(v_{s_i})$. Note that $O(k)$ time complexity is essential for a $vsplit_{new}$ transformation to update the mesh connectivity of $N(v_{s_i})$.

4.4 Degenerate Cases

When the vertices v_t and v_u have a common neighbor vertex in addition to v_l and v_r, $ecol(v_s, v_t, v_u, v_l, v_r)$ results in a mesh whose graph structure is not simply connected. Hoppe et al. considered this $ecol$ transformation to be illegal and did not allow it in a simplification process (cf. Theorem 4. in the appendix of [12]). In this paper, if this case occurs in the synthesis phase, we postpone $ecol(v_{s_i}, v_{t_i}, v_{u_i}, v_l^a, v_r^a)$ until all common neighbor vertices of v_{t_i} and v_{u_i} other than v_l^a and v_r^a disappear by vertex split transformations.

A degenerate case may also occur in a vertex split transformation in the synthesis phase. When the two cut vertices v_l^a and v_r^a are the same, $vsplit(v_{s_i}, v_{t_i}, v_{u_i}, v_l^a, v_r^a)$ operator is not well defined. If we enforce the $vsplit$ transformation in this case, the resulting mesh will not be simply connected. This situation can be resolved by performing $vsplit(v_l^a, \cdots)$ prior to $vsplit(v_{s_i}, v_{t_i}, v_{u_i}, v_l^a, v_r^a)$. Note that this additional vertex split is necessary just to maintain the simply connectedness of a selectively refined mesh, not to satisfy the preconditions given by the reserved topological details, as in previous work.

5 Experimental Results

To demonstrate the benefit of the proposed selective refinement scheme of progressive meshes, we applied the scheme to the following three types of mesh models with different refinement criteria.

- $2\frac{1}{2}$D terrain data (Figure 8)
- 2D regular mesh constructed from an image (Figure 9)
- 3D polygonal mesh models (Figures 10 and 11)

In the experiments, the error quadrics proposed by Garland and Hackbert [4, 5] were used to determine the order of edge collapses in the analysis phase.

In Figure 8, we compared the proposed scheme with previous approaches [24, 9] in a view-dependent rendering framework, where the view-frustum test was used to selectively refine the Grand Canyon terrain data. The statistics of the numbers of vertices and faces are reported in the 'Terrain1' column of Table 2. From Figure 8 and Table 2, we can see that the proposed scheme generates a selectively refined mesh with smaller numbers of vertices and faces than previous schemes. This results from the fact that the proposed scheme does not incur additional vertex splits around the selected vertices to be refined except the degenerate cases, while additional vertices may be split in previous schemes to satisfy the preconditions for the selected vertices.

Figure 9 shows the results when our and previous selective refinement schemes were applied to recover an image with a small number of triangles. A regular mesh $\hat{M}$ was obtained from the 'Lena' image of 128 $\times$ 128 size, where each vertex was centered in a pixel and assigned the pixel color. The feature points of $\hat{M}$ were selected by using the Sobel mask. Figure 9(a) shows the original image and the selected feature points. In the analysis phase, the original mesh $\hat{M}$ was simplified to a base mesh M^0 by using error quadrics with color [5]. In the synthesis phase, we selectively refined the base mesh M^0 to recover all the feature points. With this refinement criterion, all selective refinement schemes generated images with a visual quality similar to the original image, as shown in the top row of Figure 9. However, our scheme has the maximum locality in terms of the finally refined vertices, as illustrated in the bottom row of Figure 9. Again, this is due to the *truly* selective refinement property of our scheme.

Figure 10 shows an example of view-dependent refinement of a 3D mesh model generated by the proposed scheme. In this example, we adapted the view-frustum and back-face culling tests as the refinement criteria. Figure 10(a) shows the wireframe image of the face in a selectively refined mesh M and the original bunny model $\hat{M}$ with the view frustum. In Figure 10(b), the mesh M is rendered from another view direction in a wireframe image. Figure 10(c) shows the dual pieces of M overlaid on the original mesh $\hat{M}$.

Figure 11 shows an example of selective refinement around the silhouette of a horse model. In this example, we applied the silhouette test using the cone of normals [22] to each vertex. Figure 11(a) shows the resulting selectively refined mesh M. In Figure 11(b), the mesh M is viewed from a different direction. Figure 11(c) shows the dual pieces of M overlaid on the original mesh $\hat{M}$.

Table 2 summarizes the statistics of the experiments in which our and previous selective refinement schemes were applied to several meshes with different refinement criteria. From Table 2, we can see that our scheme always generates selectively refined meshes with smaller numbers of vertices and faces than previous schemes. When we apply selective refinement to an interactive application, such as a navigation system, the average time for performing $vsplit$ and $ecol$ transformations is important as well as the numbers of vertices and faces of the resulting meshes. In our implementation of the proposed scheme, $vsplit_{new}$ and $ecol_{new}$ transformations take 0.022 msec and 0.017 msec on an 866MHz Pentium III system, respectively. This speed shows that the process of finding the cut vertices in $vsplit_{new}$ and $ecol_{new}$ transformations does not introduce any strong computational overhead. Table 2 also shows the numbers of degenerate cases that happen in the experiments. The degenerate cases come from the connectivity of selectively refined meshes and cannot be avoided if we wish to preserve the simply connectedness of the meshes.

6 Conclusion and Future Work

In this paper, we presented a truly selective refinement scheme of progressive meshes. By using on-the-fly computation of active cut vertices, it is possible to apply a vertex split or edge collapse transformation to any selected vertex or edge without affecting the neighborhood in a selectively refined mesh. The concept of dual pieces provided the theoretical basis for the proposed scheme. The dual piece concept can simply be extended for the design of a truly selective refinement scheme of other progressive mesh representations, such as triangle-collapse-based approach [6].

Due to the abrupt resolution change, the proposed scheme could generate a flipped triangle in the synthesis phase. In this paper, we focused on the connectivity (topology) of a selectively refined mesh, not considering its geometry. We expect that the triangle flipping problem can be resolved by introducing a refinement criterion that prevents drastic changes of face normals.

The proposed selective refinement scheme will be useful in applications that require local and hierarchical refinement of irregular meshes, such as view-dependent simplification, mesh editing, mesh morphing, and mesh reparameterization. We hope that the proposed scheme will contribute to the expansion of the application area of progressive meshes. For instance, a progressive mesh with normal and texture maps [1, 19] can be used to efficiently generate a realistic image when it is adaptively refined at the silhouette by our refinement scheme.

For future work, we are currently investigating the theoretical aspects of the proposed scheme, such as the optimality in the number of vertices and faces for a given set of vertices to be refined.

Acknowledgments

The authors would like to thank Sungyul Choe for help with implementation and Minchoel Yoon for helpful comments on the mathematical terminology. The bunny model is courtesy of the Stanford Computer Graphics Laboratory, and the horse model is courtesy of Cyberware. Grand Canyon terrain data is obtained from The United States Geological Survey (USGS), with processing by 'Terra' of Micheal Garland. This work was supported by the Ministry of Education of Korea through the Brain Korea 21 program.

References

[1] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Apperance-preserving simplification. *ACM Computer Graphics (Proc. of SIGGRAPH '98)*, pages 115–122, 1998.

[2] Tony D. DeRose, Michael Lounsbery, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. Technical report, University of Washington, 1993. Technical Report Number 93–10–05.

[3] Jihad El-Sana and Amitabh Varshney. Generalized view-dependent simplification. *Computer Graphics Forum (Proceedings of Eurographics '99)*, 18(3):83–94, 1999.

[4] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, 1997.

[5] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. *IEEE Visualization '98*, pages 263–269, 1998.

[6] Tran Gieng, Bernd Hamann, Kenneth I. Joy, Greg L. Schussman, and Issac J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Trans. Visualization and Computer Graphics*, 4(2):145–161, 1998.

[7] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '99)*, pages 325–334, 1999.

[8] Hugues Hoppe. Progressive meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '96)*, pages 99–108, 1996.

[9] Hugues Hoppe. View-dependent refinement of progressive meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, 1997.

[10] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proceedings of Visulaization '98*, pages 35–42, 1998. IEEE Computer Society Press.

[11] Hugues Hoppe, Tony DeRose, Tom Dunchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *ACM Computer Graphics (Proc. of SIGGRAPH '93)*, pages 19–26, 1993.

[12] Hugues Hoppe, Tony DeRose, Tom Dunchamp, John McDonald, and Werner Stuetzle. Mesh optimization. Technical Report TR 93-01-01, University of Washington, 1993.

[13] Donald E. Knuth. *Sorting and Searching*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[14] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '98)*, 1998.

[15] Peter Lindstrom. Out-of-core simplification of large polygonal models. *ACM Computer Graphics (Proc. of SIGGRAPH 2000)*, pages 259–262, 2000.

[16] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. *ACM Computer Graphics (Proc. of SIGGRAPH '96)*, pages 109–118, 1996.

[17] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. *IEEE Trans. on Visualization and Computer Graphics*, 6(1):79–93, 2000.

[18] Rĕmi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum (Proc. of Eurographics '96)*, 15(3):67–76, 1996.

[19] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. *ACM Computer Graphics (Proc. of SIGGRAPH 2000)*, pages 327–334, 2000.

[20] William J. Schroeder. A topology modifying progressive decimation algorithm. *Proceedings of Visulaization '97*, pages 205–212, 1997. IEEE Computer Society Press.

[21] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '92)*, pages 65–70, 1992.

[22] Leon A. Shirman and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. *Computer Graphics Forum(Proceedings of Eurographics '93)*, 12(3), 1993.

[23] Greg Turk. Re-tiling polygonal surfaces. *ACM Computer Graphics (Proc. of SIGGRAPH '92)*, pages 55–64, 1992.

[24] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. *IEEE Visualization '96*, pages 327–334, 1996.

[25] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, pages 259–268, 1997.

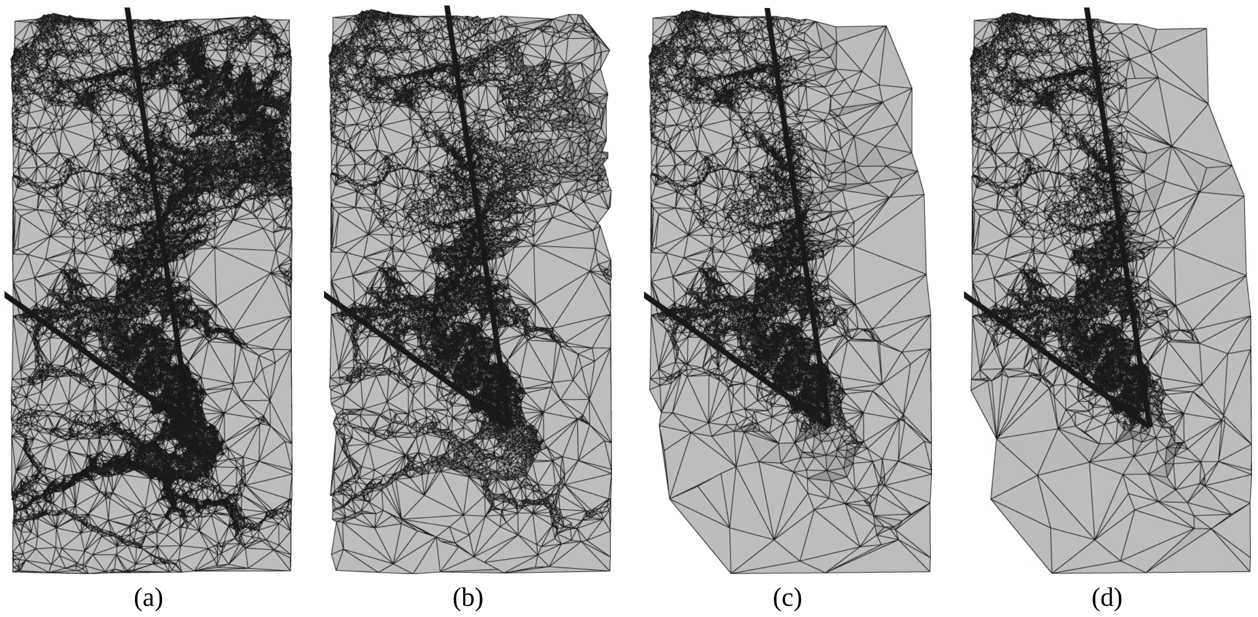


Figure 8: *View-dependent refinement: (a) terrain data; (b), (c), (d) selectively refined meshes with the schemes of [24], [9], and ours, respectively. The view-frustum is denoted by bold lines*

Model		Lena	Female	Terrain1	Terrain2	Terrain3	Cow	Horse	Bunny
Type		2D	2D	2.5D	2.5D	2.5D	3D	3D	3D
Refinement Criteria		F.P.	F.P.	V.F.	V.F.	V.F.	V.F. & B.F.C.	V.F. & B.F.C.	V.F. & B.F.C.
Original Mesh	#V	16,384	11,008	30,000	30,000	30,000	19,851	19,851	34,834
	#F	32,258	21,590	59,856	59,856	59,856	39,698	39,698	69,451
Xia & Varshney [24]	#V	10,617	5,521	14,218	3,510	7,779	15,144	15,144	8,220
	#F	20,871	10,876	28,327	6,941	15,496	30,284	30,284	16,413
Hoppe [9]	#V	7,335	3,243	11,504	1,216	5,341	12,049	12,049	4,768
	#F	14,437	6,395	22,934	2,404	10,567	24,094	24,094	9,527
Our scheme	#V	4,725	2,257	10,130	999	5,128	10,870	10,870	4,231
	#F	9,291	4,451	22,105	1,978	10,234	21,736	21,736	8,453
	#DC	6	4	52	2	22	20	68	21

Table 2: *Statistics of the experiments with several meshes and different refinement criteria (F.P.: feature points, V.F.: view-frustum, B.F.C.: back-face culling, #V: the number of vertices, #F: the number of faces, #DC: the number of degenerate cases)*

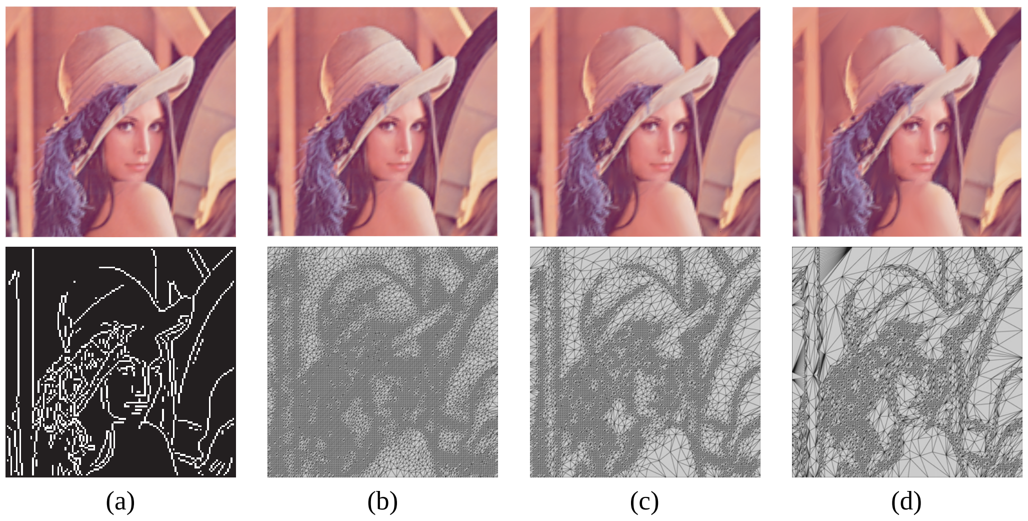


Figure 9: Feature-based selective refinement: (a) original image (128×128 *vertices) and its feature points; (b), (c), (d) selectively refined meshes to recover the feature points with the schemes of [24], [9], and ours, respectively. The images of the top row were obtained by Gouraud shading.*

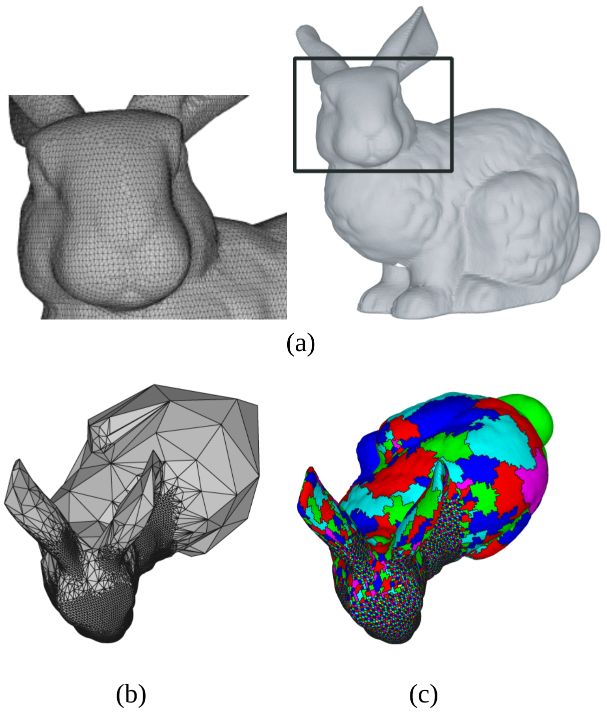


Figure 10: View-dependent refinement of the bunny model: (a) close-up view of the face and the original model; (b) wireframe image of the selectively refined mesh from a different view direction; (c) corresponding dual pieces of (b) over the original model

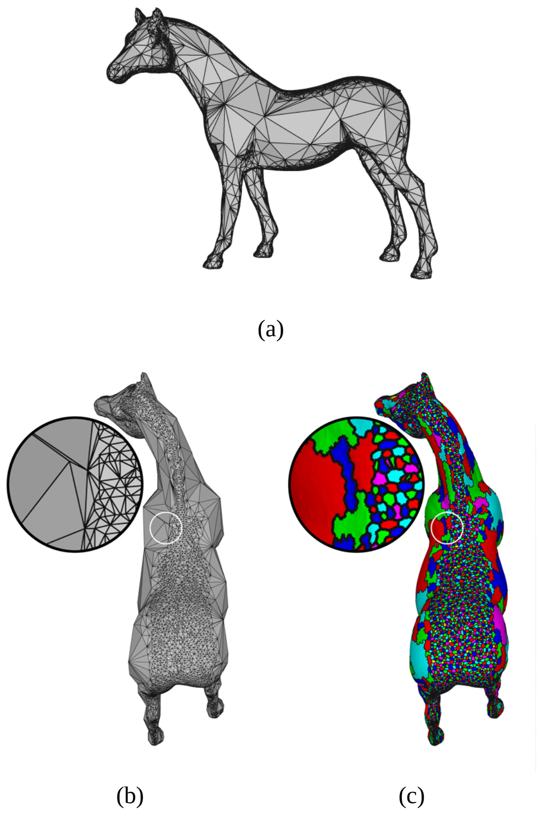


Figure 11: Silhouette refinement of a horse model: (a) selectively refined mesh; (b) wireframe image of (a) from another view direction; (c) corresponding dual pieces overlaid on the original mesh